Projet d'Initiative Personnelle

Institut Supérieur de l'Aéronautique et de l'Espace



Pilote automatique en JAVA

Alexandre FERNANDES
Hugo MERIC
Guillaume SMITH

Tuteur : Fabrice FRANCES

Remerciements

Nous tenons à remercier toutes les personnes qui nous ont aidé à mener à bien ce projet, notamment M. Mouton pour son aide lors de la réalisation du châssis et M. Francès pour la partie informatique.

Table des Matières

Chapitre 1

Présentation du projet

1.1 INTRODUCTION

Le DMI a acheté l'an dernier deux Pob-Bot, qui sont des robots programmables en java. Plusieurs fonctionnalités étaient déjà présentes sur ces robots comme par exemple le suivi de formes prédéfinies. Après avoir lu un article sur internet concernant les pilotes automatiques de voiture, nous avons choisi d'utiliser un de ces robots pour notre Projet d'Initiative Personnelle. L'objectif fixé est de faire faire au robot des tours de pistes sur le circuit de la Ramée à Toulouse Figure 1.1.



Figure 1.1 Circuit de La Ramée

Cependant, le robot ne doit pas suivre les lignes délimitant le circuit car cela a déjà été réalisé en projet d'électronique numérique de deuxième année et ne nécessite aucune programmation. Ainsi le Pob-Bot doit être capable de détecter la présence de ligne ou non, et ensuite d'agir en conséquences.

1.2 LES DIFFÉRENTES PARTIES DU PROJET

Tout d'abord, il a fallu modifier le châssis du robot. En effet, comme nous le verrons dans la deuxième partie, le Pob-Bot est monté sur des chenilles ce qui entraîne plusieurs limitations au niveau de son comportement. Une fois le nouveau châssis monté, il a fallu alors s'intéresser aux algorithmes de reconnaissance de lignes. De nombreux algorithmes existent mais ils ne s'appliquent pas forcément à notre projet. C'est pourquoi il a d'abord fallu récupérer un échantillon d'images que voient le robot sur la piste. Après avoir ces images, nous avons pu tester les algorithmes existants et aussi développer nos propres algorithmes. La dernière partie a consisté à implémenter ces algorithmes sur le robot et ensuite de les faire tourner pour voir comment celui-ci se comporte. L'implémentation des algorithmes n'a pas été une partie de plaisir car comme tout système embarqué, le Pob-Bot a de nombreuses limitations: mémoire, compilation... De plus, une API presque inexistante ne facilite pas les choses ainsi que quelques autres surprises que nous détaillerons plus tard.

1.3 DESCRIPTIF DU MATÉRIEL UTILISÉ

Nous allons décrire ici les caractéristiques du Pob-Bot. Comme nous l'avons dit précédemment, le Pob-Bot est un robot mobile programmable Figure 1.2.



Figure 1.2 Pob-Bot

Le Pob-Bot est composé d'un Pob-Eye Figure 1.3, une caméra couleur intelligente, et d'un Pob-Proto qui est la carte des commandes d'entrées et sorties Figure 1.4.

Le Pob-Eye est le coeur du robot. C'est lui que nous programmons en java en utilisant Pob-Java, logiciel fourni avec le robot. Le Pob-Eye dispose de 64 Ko de RAM. 32 Ko de RAM sont réservés au stockage d'une image de la caméra (on stocke les 3 couleurs). Seulement 27 Ko de RAM sont disponibles pour notre programme car 5 Ko sont utilisés pour l'environnement java. Il faut donc être vigilant sur l'utilisation des objets java. Les images de la caméra ont une résolution de 88*120 pixels.

La carte Pob-Eye peut fonctionner en deux modes: programmation et exécution. Le mode programmation permet, en reliant la carte au PC via le port série, de charger nos programmes sur le robot à l'aide de Pob-Java. Une fois le chargement terminé, on passe en mode exécution et dès lors le programme chargé est exécuté par la carte.

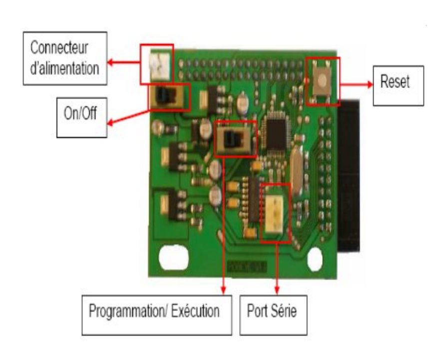


Figure 1.3 Carte Pob-Eye

Le Pob-Bot comporte aussi un Pob-Proto qui est un périphérique destiné au Pob-Eye. Le Pob-Proto est équipé de:

- 6 connecteurs pour commander des servomoteurs.
- Un joystick analogique et son bouton poussoir.

La valeur de consigne des connecteurs pour piloter les servomoteurs est comprise entre 0 et 255. Pour notre projet, nous utilisons seulement la plage comprise entre 90 et 230 par limitation physique du châssis utilisé. Le joystick analogique nous a servi pendant la phase de tests de nos algorithmes (pour lancer l'acquisition d'une image et son traitement).

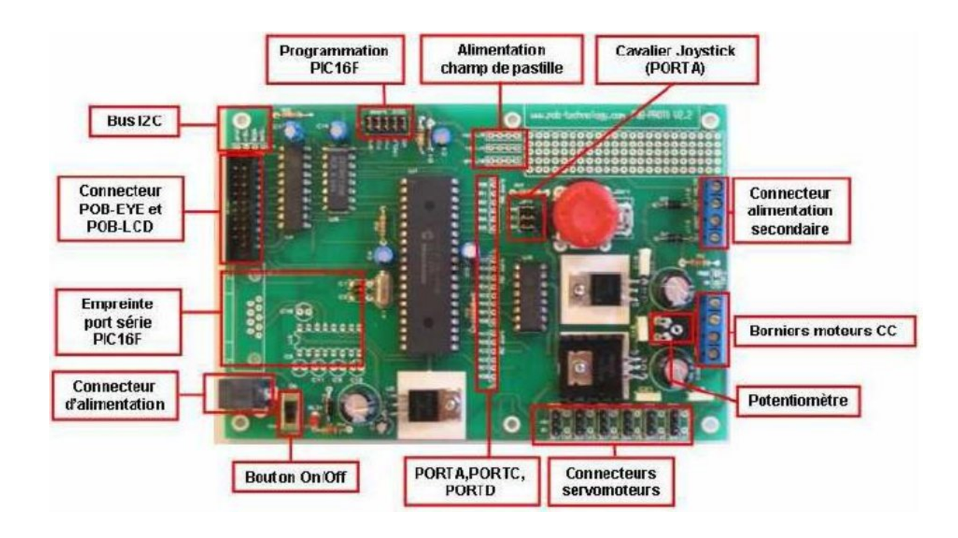


Figure 1.4 Carte Pob-Proto

1.4 MODE D'EMPLOI DE POB-JAVA

Pob-Java est un outil pour programmer, charger et débuguer le module Pob-Eye en java. Cet outil est constitué de 6 onglets ayant chacun une fonction particulière Figure 1.5.

On commence par écrire notre programme dans un bloc-notes. Pour faciliter la lisibilité, nous avons choisi d'utiliser Notepad++ pour coder.

L'onglet Pob-Project permet de sélectionner le projet que l'on souhaite charger sur la carte. Le Pob-Bot doit être relié au PC via un port série. Nos ordinateurs portables n'ayant pas de port série, nous avons utilisé un câble USB-série fourni par notre tuteur. Il faut aussi choisir le numéro de port sur lequel le Pob-Eye est connecté au PC.

L'onglet Pob-compiler sert à compiler l'application en un langage compris par le Pob-Eye. Il faut donner un nom pour le fichier de sortie que va générer le compilateur et donner le fichier principal java (classe qui contient la méthode main()). Si le code ne comporte pas d'erreurs, l'application est correctement compilée et on peut passer à l'onglet suivant. Sinon la console de Pob-Java indique en général où se trouve les erreurs dans le code.

L'onglet Pob-loader charge l'application dans le module Pob-Eye. Au moment du chargement, le Pob-Eye doit être en mode programmation et le bouton reset a du être pressé. Une fois l'application chargée, il faut éteindre et rallumer le Pob-Proto en mode exécution pour lancer l'application chargée sur la carte.

Nous n'avons pas utilisé les onglets Pob-Bitmap et Pob-Pattern pour notre projet. Ces onglets gèrent le dictionnaire de forme que peux reconnaître le Pob-Bot et l'affichage de formes sur l'écran LCD.

Le Pob-Terminal aide au débugage de l'application via le lien série du Pob-Eye. En effet, dans notre code on peut demander au robot d'écrire dans la console à l'aide de la commande System.print(«test»), équivalent du System.out.println(«test») sous java classique. Ceci aide à savoir jusqu'où le programme est allé lors de son exécution.



Figure 1.5 Pob-JAVA

Chapitre 2

Modification du châssis

2.1 CHÂSSIS DU POB-BOT

Comme nous l'avons déjà vu précédemment, le Pob-Bot était initialement monté sur des chenilles. Cela avait plusieurs conséquences sur le comportement du robot.

Tout d'abord, le robot ne pouvait pas tourner en étant en mouvement. Il devait s'arrêter, tourner et repartir. De plus, la vitesse du robot était alors très limité. Tout cela n'était pas compatible avec notre projet. Nous avons donc choisi de changer de châssis. Nous nous sommes donc rendus au DAS pour demander conseil à Bernard Mouton. Au vu de nos besoins, M Mouton nous a conseillé d'acheter une voiture dans un hypermarché car cela convenait parfaitement à nos exigences et revenait beaucoup moins cher que si nous nous adressions à un magasin de modélisme car nous ne désirions pas une voiture trop rapide vu les capacités du Pob-Eye (la caméra ne peut prendre que 8 images par seconde).

2.2 MONTAGE DU 1ER CHÂSSIS

Nous avons donc acheté une première voiture. Après l'avoir démontée, on a découvert que la direction n'utilisait pas de servomoteur. Cela ne posait pas de réels problèmes mais limitait l'utilisation du robot car il n'avait alors que trois positions de direction sans intermédiaire (gauche, droite et tout droit).

Ensuite, il a fallu monter le Pob-Eye et le Pob-Proto sur le châssis. Nous avons donc réalisé des plans sous Catia pour que l'atelier nous réalise une plaque en aluminium que nous allions monter sur le châssis Figure 2.1. L'aluminium avait l'avantage d'être très léger et solide donc de ne pas gêner le Pob-Bot en mouvement (pas de surcharge pour le moteur).

La plaque permet aussi de fixer la caméra à des hauteurs différentes. Cela nous a permis par la suite de régler la caméra à une hauteur convenable pour ne pas voir trop loin car dans ce cas elle observe le paysage en dehors de la piste: bâtiments, arbres... Cependant,

la caméra devait voir assez loin pour être capable de tourner entre le moment où elle repère une ligne et le temps que l'algorithme de reconnaissance commande le servomoteur.

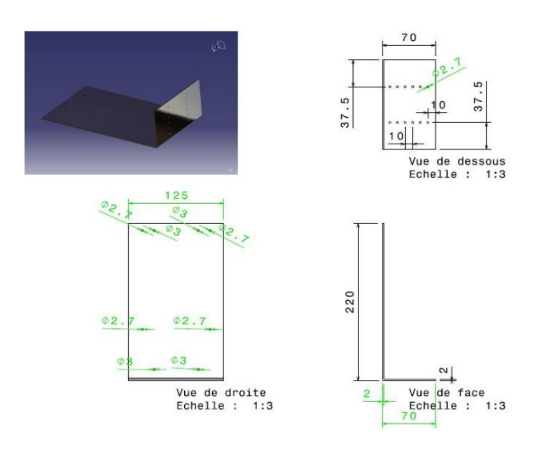


Figure 2.1 Modèle catia de la plaque

Malheureusement, lors des premiers tests, nous avons rencontré des problèmes avec le moteur. En effet, il nécessitait 0.5 A pour être alimenté ce qui faisait chauffer la carte de la voiture achetée. Nous avons donc dû changer de châssis car le circuit de celui-ci avait trop chauffé et s'était détérioré, ce qui rendait le châssis inutilisable.

2.3 MONTAGE DU CHÂSSIS FINAL

Pour ce nouveau châssis, nous avons acheté une voiture un peu plus chère mais qui avait l'avantage d'avoir un servomoteur pour contrôler la direction et nous pouvions aussi utiliser notre plaque en aluminium.

Le servomoteur de la voiture possédait un câble électrique à 6 fils ce qui n'est pas habituel et qui ne peut pas s'adapter sur le Pob-Proto (3 fils seulement). Le DAS dispose cependant de nombreux servomoteurs, M Mouton nous en a fournit un que nous avons adapté pour notre voiture.

Lorsque l'on a installé la plaque, nous l'avons incliné un peu en avant à l'aide de fixations plus longues à l'arrière. Cela avait pur but de réduire le champs de vision lointain de la caméra et donc de se concentrer sur la piste.

La propulsion de la voiture se gère avec la télécommande fourni en achetant la voiture. Nous avons fait ce choix pour éviter de couper des fils au niveau des câbles du servomoteur contrôlant l'avancée car il fallait utiliser la batterie de la voiture pour fournir de la puissance, que le Pob-Bot ne peut pas forcément fournir. Cela nous permet aussi de pouvoir l'arrêter si besoin (sortie de piste si ligne non détectée...). La direction est gérée par le servomoteur directement reliée au Pob-Proto. Des tests sur le servomoteur ont montré que les consignes de commande varient entre 90 (à fond à gauche) et 230 (à fond à droite). Sachant que droit devant correspond à une consigne de 160.

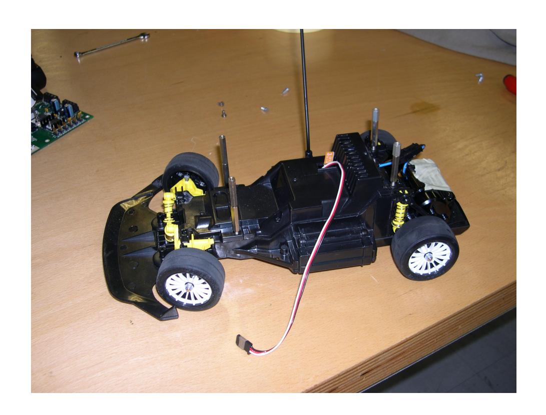


Figure 2.2 Châssis avec le fil qui sert à commander le servomoteur

L'alimentation de la carte se fait à l'aide de 8 piles monté sur le châssis. Pour éviter que les piles ne bougent pendant que le Pob-Bot roule, nous les avons fixé sur la plaque en aluminium à l'aide de velcro.

Pour finir, nous avons remarqué que pendant que le robot roulait, le moteur situé à l'arrière de la voiture venait traîner contre le sol. Cela était dû aux suspensions arrières qui n'étaient pas assez rigides. Nous n'avons pas trouvé de ressorts au DAS pour changer nos amortisseurs. Ainsi, on a décidé de scotcher le moteur pour qu'il ne traîne plus pendant les manoeuvres.

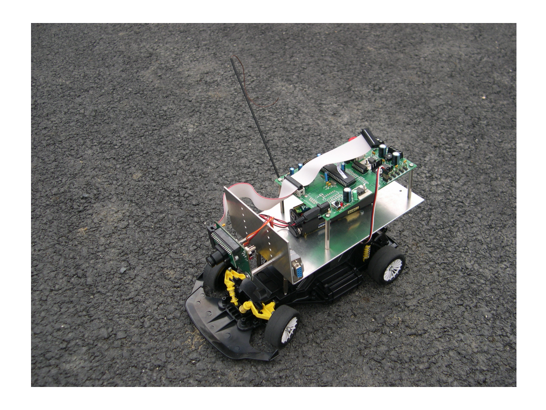


Figure 2.3 Châssis final

Chapitre 3

Algorithmes de reconnaissance de lignes

Le montage du châssis terminé, il a fallu étudier les algorithmes de reconnaissance de lignes qui existaient déjà. Plusieurs algorithmes ont déjà été développés dans des cadres plus généraux et il a donc été nécessaire de tester ces algorithmes sur les images prises par le Pob-Bot à la Ramée. Nous allons donc voir dans un premier temps deux algorithmes souvent utilisés pour la détermination de droites ou de contours. Ensuite, nous allons étudier deux algorithmes que nous avons développé en fonction de nos besoins et de ce que nous avions trouvé pendant nos recherches.

3.1 ALGORITHMES DÉJÀ EXISTANTS : HOUGH ET CANNY

3.1.1 Transformée de Hough

La transformée de Hough est une technique de reconnaissance de forme inventée en 1962 par Paul Hough. L'application la plus simple permet de reconnaître les lignes d'une image, mais des modifications peuvent être apportées pour reconnaitre n'importe quelle forme. Le code de l' algorithme de base (reconnaissance de lignes) est disponible en annexe.

Un petit logiciel permettant de charger des images et d'appliquer la transformée de Hough nous a permis de tester l'efficacité de celle-ci sur les images prises par le Pob-Eye. Voici ce que nous avons obtenu sur plusieurs images:

On remarque alors que la transformée de Hough arrive à chaque fois à trouver la ligne. Cependant, la ligne est perdu entre de nombreuses lignes parasites. Certaines sont dues à l'image prise par le Pob-Eye (par exemple on remarque une ligne verticale toujours présente sur la droite de l'écran). D'autres lignes sont dues au paysage en dehors de la piste qui créent des lignes (arbres, bâtiments...). On peut alors essayer de couper l'image en deux selon l'horizontale et ne s'intéresser qu'à la moitié du bas. Cependant, l'algorithme trouve toujours des droites même quand l'image observée ne contient pas de lignes.

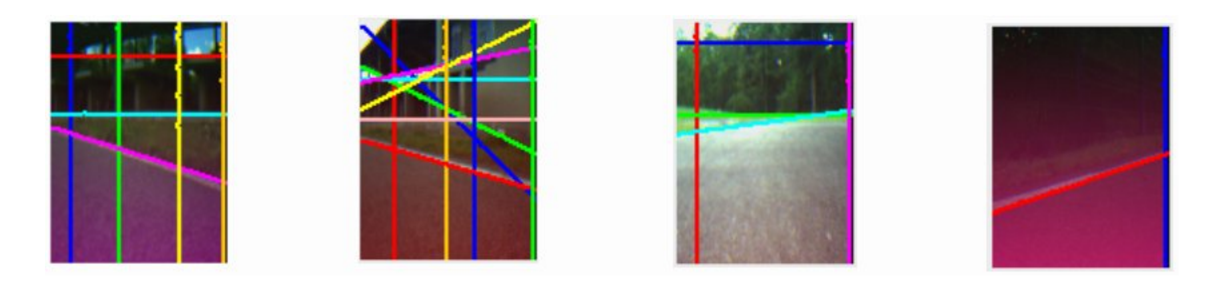


Figure 3.1 Transformée de Hough

On peut aussi modifier la sensibilité de la transformée pour réduire le nombre de lignes trouvées par l'algorithme. Cependant, cela ne permet pas d'éliminer toutes les lignes parasites et on peut aussi perdre certaines lignes qui délimitent la piste. On se rend alors compte que la transformée de Hough n'est pas très efficace pour notre problème.

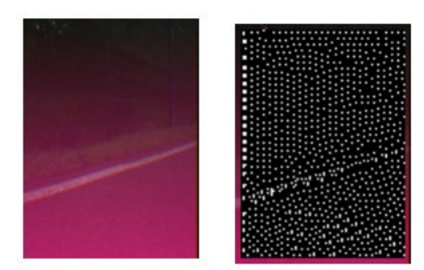
De plus, on se rendit compte par suite que les limitations mémoire du Pob-Bot ne permettent pas d'implémenter correctement cet algorithme sur le robot. En effet, la transformée de Hough nécessite pas mal de calculs sur des flottants et d'espace mémoire.

3.1.2 Filtre de Canny

L'algorithme de Canny est utilisé en analyse d'images pour la détection des contours. Cet algorithme est basé sur une méthode de gradient. On cherche à garder les pixels qui participent aux contours . En terme de gradient, cela signifie un «pic» dans la variation d'intensité. Pour détecter un «pic», on regarde les pixels voisins, et s'ils ont tous un gradient plus faible que le pixel observé, alors le pixel observé est un «pic».

Le gradient se calcule en chaque pixel en utilisant l'opérateur de Sobel qui est composé de deux matrices 3*3 (une par direction spatiale). Nous ne développons pas ici les calculs du filtre de Canny car ils ne présentent pas d'intérêts en eux-mêmes. Cependant, le code est donné en annexe.

L'algorithme de Canny n'a pas donné de résultats très concluants sur la reconnaissance de ligne,





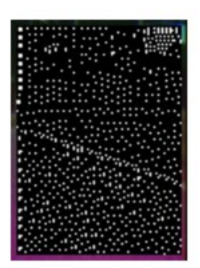


Figure 3.2 Filtre de Canny

On arrive à deviner les lignes à l'oeil mais on ne peut rien obtenir sur le Pob-Bot. Nous avons alors développé nous-même deux algorithmes pour essayer de mieux s'adapter à notre problème, celui-ci étant très particulier (nous savons que nous recherchons une ligne blanche sur du bitume).

3.1.3 Algorithme 1

Le premier algorithme utilise un filtre pour récupérer le maximum de points sur la ligne avant de faire deux régressions linéaires pour obtenir la ligne.

Après que le Pob-Eye ait récupéré l'image, on commence par couper l'image en deux selon l'horizontale. En effet, on a remarqué que la moitié supérieure de l'image ne comprenait que peu d'informations intéressantes (paysage, fin de lignes...).

Ensuite, on balaye l'image pour calculer le maximum et le minimum d'intensité en utilisant seulement les composantes bleues et vertes de chaque pixel car la caméra avait tendance à voir rouge en extérieur.




Figure 3.3 Limite de la caméra

Une fois le calcul réalisé, on calcule le rapport $q = \frac{max-min}{max}$ qui va nous servir à filtrer l'image. Si q est trop faible, alors on ne traite pas l'image car il n'y a pas assez de contraste. On effectue cela car nous savons que nous recherchons une ligne blanche sur du bitume plutôt foncé, ce qui induit un fort contraste. Dans le cas contraire, on balaye de nouveau l'image et pour chaque pixel, on teste si la somme de ces composantes vertes et bleues est supérieure à un coefficient α dépendant de q et du maximum d'intensité. Si la somme est supérieure, alors le pixel devient blanc, sinon noir. Le coefficient α est de la forme:

$$\alpha = (1 - 0.3q^2)max$$

Par exemple, si le contraste est fort, alors q est faible et on ne garde que les points dont la somme des composantes est proche du max. Le coefficient 0.3 a été déterminé expérimentalement à partir de toutes les photos que nous avions faites à la Ramée.

Si la proportion de points blancs sur la demi-image est trop élevée, alors on ne fait rien. En effet, cela a souvent lieu quand le Pob-Eye ne voit pas de lignes. Il observe dans ce cas seulement la piste. On s'imagine que dans ce cas q va être suffisamment faible et on ne va pas traiter l'image mais ce n'est pas toujours le cas, le soleil peut parfois entraîner un contraste élevé dû aux reflets sur le bitume et on récupère alors beaucoup de points blancs.

Figure 3.4 Etape 1: division de l'image et transformation en noir et blanc

L'image est maintenant en noir et blanc. On commence par enlever les pixels blancs isolés. Ensuite, on va récupérer seulement certains points blancs pour ensuite faire les régressions linéaires. Il est nécessaire de récupérer seulement quelques points car on peut avoir des points isolés qui ne contiennent donc pas d'informations. Pour sélectionner un point, on considère le nombre de points blancs se trouvant à proximité et sur la même colonne. S'il y a plus de 2 points blancs chacun espacés d'au plus 1 pixel noire sur une colonne, alors on prend le milieux de ceux-ci et on considère ce milieu comme un point de la ligne. La Figure 3.5 explique la sélection des points utilisés pour la régression linéaire.

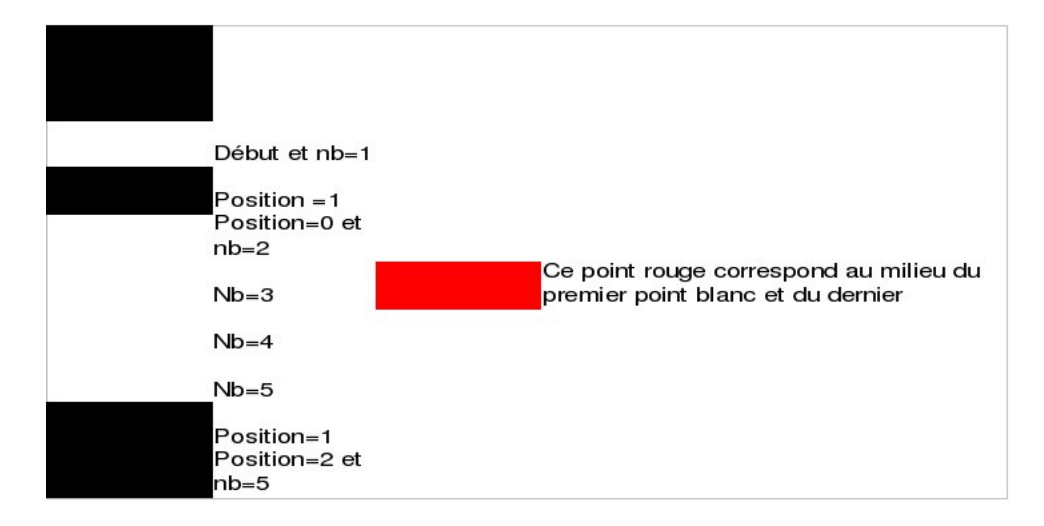


Figure 3.5 Algorithme de sélection des points pour la régression linéaire

Après avoir récupéré ces points Figure 3.6, on réalise une première régression linéaire. On obtient alors une première droite. Pour améliorer la précision, on élimine les points qui ont servi à la première régression linéaire et qui se situent trop loin de la droite obtenue. On réalise alors une autre régression pour obtenir la droite finale (y=ax+b).

En fonction des coefficients a et b obtenus, on peut alors demander au robot de tourner. La Figure 3.7 présente quelques résultats obtenus sur des images de la Ramée où les points rouges représentent les points gardés pour la régression linéaire. En bleu, on a la droite obtenue avec la régression.



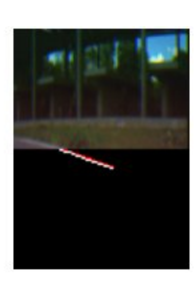
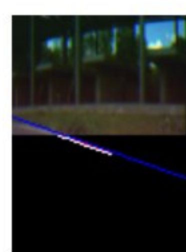



Figure 3.6 Exemple d'ensemble de points retenus pour la régression linéaire







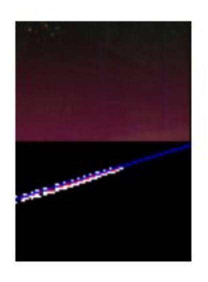



Figure 3.7 Résultats de l'algorithme 1

3.1.4 Algorithme 2

Le deuxième algorithme détermine les contours de la piste à partir d'une valeur d'intensité moyenne calculée à partir de l'image.

Pour calculer la valeur moyenne de l'intensité, on ne prend en compte que les composantes bleues et vertes des pixels pour les mêmes raisons que précédemment. On choisit deux petits carrés de 3 pixels de largeur en bas à gauche et à droite de l'image. Pour chaque carré, on calcule la moyenne de l'intensité des pixels qui le composent. On obtient alors deux valeurs moyennes dont on fait encore la moyenne.

Après avoir calculé cette valeur, on parcourt l'image en partant d'en bas à gauche et en remontant verticalement. Dès qu'un pixel à une intensité supérieure à la valeur moyenne plus une constante (fixée à 40 ici), on sélectionne ce pixel comme un point du contour et on passe à la colonne suivante. Si on ne trouve pas de pixel appartenant au contour à la moitié de l'image, on n'a pas de points du contour dans la colonne parcourue et on passe à la suivante.

Une fois l'image parcourue, on fait une régression linéaire sur les poins obtenue comme pour l'algorithme 1. Les résultats sont un peu moins précis que précédemment mais permettent de donner une commande de direction au robot. La Figure 3.8 montrent les résultats obtenus sur les photos prises à la Ramée par le Pob-Eye.

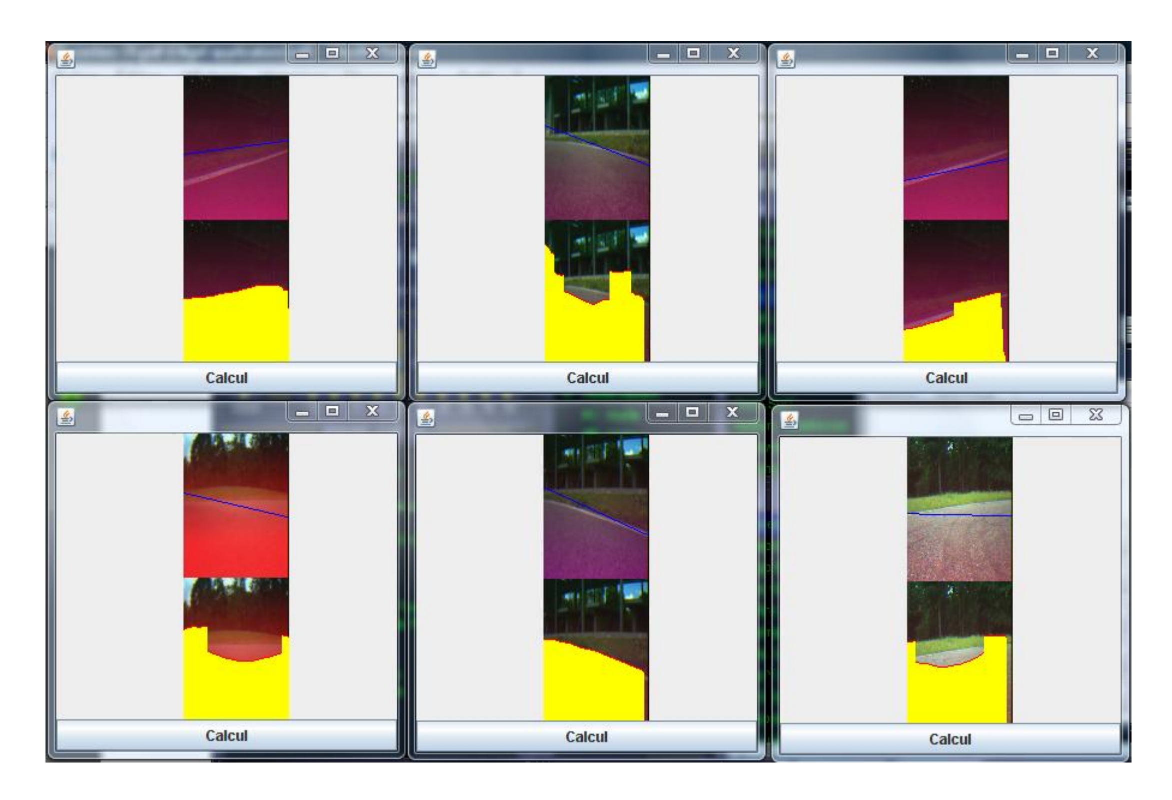


Figure 3.8 Résultats obtenus avec l'algorithme 2

Chapitre 4

Programmation du Pob-Bot

Après avoir monté le châssis sur lequel se trouve le matériel, et après avoir conçu des algorithmes de reconnaissance de lignes sous Eclipse, il faut charger nos algorithmes sur le Pob-Bot.

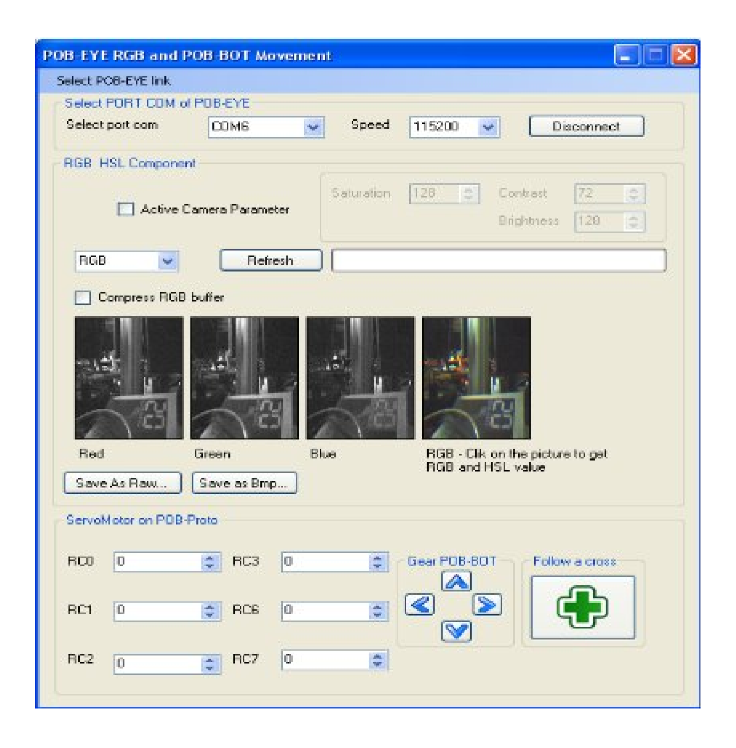
Pob-Java permet de charger des programmes écrit en java sur la carte Pob-Eye. Mais cette carte ne supporte qu'un java assez simple: il n'y a pas de liste, de matrice... Il a donc fallu modifier les algorithmes réalisés dans la partie précédente afin qu'ils puissent être compilés sur la carte.

4.1 PRISE EN MAIN DU POB-BOT

Pour commencer, nous avons effectué quelques tests pour nous familiariser avec Pob-Java et le Pob-Bot. Il est à noter qu'une API est fournie avec le robot. Seulement celle-ci est très succincte et comporte même des erreurs. Par exemple, le temps à mettre dans un System.wait() n'est pas en milliseconde mais en microseconde. Cela change beaucoup de choses sur le comportement du robot: demandez lui par exemple de tourner ses roues vers la gauche toutes les 1000 microsecondes en pensant que vous lui demander de les tourner toutes les secondes...

Une première chose a été d'observer la portée de la caméra et à quelle hauteur la monter sur le châssis. Cela nous a permis d'apprendre à utiliser Pob-Tools qui est un logiciel fourni avec le robot ayant le même rôle que Pob-Java mais pour un autre langage. Un programme permettant de visualiser ce que voit la caméra est disponible dans la documentation du Pob-Bot (il se trouve dans le dossier programme de la documentation, c'est le dossier RgbAndServo. Il faut alors charger une application sur le Pob-Bot et ensuite utilisé le programme RgbAndServo se trouvant dans le sous-dossier windows).

Nous avons donc charger ce programme sur le Pob-Eye pour régler la hauteur de la caméra. Après des essais réalisés à l'ENSICA, nous avons placé le Pob-Eye le plus bas possible et



légèrement incliné vers le bas car il s'avère que celui-ci possède une bonne vision et voit donc bien au-delà de la piste.



Figure 4.1 Test de la portée à l'ENSICA

Après cela, nous somme passé à la réalisation d'un échantillon d'images pour ensuite tester nos algorithmes. Le programme précédent nous a encore servi car il permet non seulement de visualiser ce que voit le Pob-Eye, mais aussi de sauvegarder sur l'ordinateur les images observées. A chaque acquisition, on obtient 4 fichiers bmp: une image RGB et une pour chaque composante RGB. La réalisation de cette bibliothèque a permis aussi de valider la position du Pob-Eye sur le châssis.

Ensuite, nous avons appris à utiliser les servomoteurs. Tout ceci peut paraitre simple, mais il y a énormément de petites astuces qui ne sont pas expliquées dans la documentation

fournie avec le Pob-Bot, comme le fait qu'après un chargement d'un programme, il faut passer en mode exécution, éteindre la carte et enfin la rallumer pour que l'algorithme se lance. Le simple passage en mode exécution sans éteindre la carte ne suffit pas.

Le contrôle des servomoteurs se fait en utilisant la classe Bot fournit avec le robot (cette classe est à récupérer dans des exemples). La classe fournit une méthode static setHead(int n) qui permet de commander le servomoteur et donc la position des roues.

On peut aussi gérer le joystick de la carte Pob-Proto avec la classe Pad fournit par les développeurs. Cela nous a permis de réaliser des tests sur les servomoteurs. Quand on pousse le joystick vers le haut, on tourne à gauche et inversement quand on pousse le joystick vers le bas.

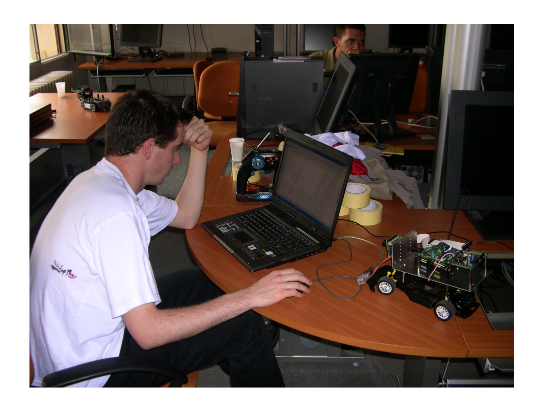


Figure 4.2 Séance PIP à la plateforme

4.2 IMPLÉMENTATION DE NOS ALGORITHMES

Nous avons donc tout d'abord modifier nos matrices en tableaux (les lignes des matrices sont mises bout à bout). Ensuite, il nous a fallu changer tous les doubles en float. Et bien entendu, nous avons dû changer le mode d'acquisition de l'image: précédemment, nous prenions une image se trouvant sur le disque dur de l'ordinateur, maintenant il faut récupérer l'image directement à partir de la Pob-Eye.

Lors d'une acquisition d'image, la Pob-Eye crée 3 tableaux différents, ils représentent les images en rouge, vert et bleu. Ensuite, nous pouvons récupérer pour chaque pixels ces 3 composantes RGB. Comme vu précédemment, nous ne prenons pas les composantes rouges à cause d'une imperfection de la caméra, qui pourrait induire des erreurs.

4.2.1 Algorithme 1

Pour le premier algorithme, nous avons rencontré de nombreux problèmes à cette phase. En effet, pour se rapprocher de ce que nous avions fait sous Eclipse, nous avons créé un tableau dans lequel nous stockions l'image. Ce tableau faisait une taille de 5280 cases. Nous avons compris après plusieurs heures de tests dans les boucles (à l'aide de System.print()

) que le Pob-Bot accordait une place mémoire au tableau qui devait représenter l'image (le tableau était bien créé), mais dès qu'on voulait y mettre une valeur, l'algorithme s'arrêtait. Ceci est sûrement du à la place mémoire très limité du Pob-Bot qui ne permet pas de créer un tableau de plus de 5000 éléments (nous n'avons pas effectué un test de limite inférieur pour déterminer la capacité de mémoire du POB-Bot).

Nous avons donc modifié l'algorithme pour ne plus créer ce tableau. Mais ceci nous a forcé à supprimer la méthode supprimerVoisins() car on ne peut pas modifier l'image en mémoire dans la Pob-Eye et on ne pouvait pas non plus l'enregistrer dans un tableau, ce qui nous a beaucoup limité dans la programmation.

Il ne nous restait donc qu'un tableau de points sur lequel nous effectuons la régression linéaire (les points rouges sur les images de la partie précédente). Ensuite, on supprimait les points trop éloignés pour effectuer une seconde régression linéaire. Ces points ont du être sauvegardé dans deux tableaux (les abscisses et les ordonnés). Il a donc fallu faire attention de ne pas dépasser la capacités des tableaux car on ne pouvait pas avoir un tableau de trop grande taille vu le problème précédent. Nous avons fixé arbitrairement leurs tailles à $172 = 2 \times 88$ (la largeur de l'image fait 88 pixels). Il a fallu faire attention au remplissage de ces tableaux pour ne pas prendre trop de points lors du traitement de l'image, et aussi de ne pas utiliser les cases du tableau qui ne correspondent pas à des points de la régression actuelle (points des régressions précédentes non modifiés).

La limitation en mémoire nous a aussi obligé à mettre en attribut tous les entiers, flottants et tableaux que nous utilisions pour éviter de les créer à chaque traitement d'image et donc de saturer très vite la mémoire du robot. En effet, nous avons pu observer une fuite de mémoire qui pouvait bloqué le Pob-Bot au bout d'un certain temps d'utilisation. Ainsi nous utilisons seulement deux tableaux pour stocker les coordonnées des points utilisés pour la régression linéaire.

Par contre, il y a des problèmes que nous avons observé et qui restent incompréhensibles: nous n'avions plus de double, mais en compilant le programme, Pob-Java en trouvait encore et donc ne voulait pas compiler. En modifiant le code, ce problème s'est résolu sans qu'on sache réellement pourquoi.

4.2.2 Algorithme 2

En parallèle, nous avons aussi modifié le second algorithme pour pouvoir le compiler avec Pob-Java et le charger sur le Pob-Eye.

Dans cet algorithme, il n'y a pas eu le problème du tableau image, mais il a aussi fallu faire attention au double. Ensuite, il y a un problème vu dans l'algorithme précédent qui n'avait pas lieu dans celui-ci: la taille des tableaux des points de ligne. En effet, ici, il y en a exactement 88 (la largeur de l'image). Sa taille est donc assez petite et totalement déterminée.

4.2.3 Résultats

Les premiers résultats ont été assez long à venir. Tout d'abord, cela venait de la difficulté de générer un code compilable pour le Pob-Bot et qui donnait de bons résultats. Cependant, de petites erreurs de notre part nous ont fait perdre beaucoup de temps. Par exemple, une division qui n'avait pas lieu d'être dans la régression linéaire et qui n'engendre pas de problèmes lors de la compilation nous a retenu assez longtemps. Après plusieurs modifications de nos algorithmes, nous avons eu notre premier résultat concluant: le second algorithme a reconnu une ligne faite avec du papier sur de la moquette bleue foncée. Les roues de la voiture tournaient bien dans la direction voulue (on choisissait cette direction uniquement suivant la pente de la ligne observée). Mais le calcul s'avérait beaucoup trop long dans certains cas (pratiquement 2 secondes).

Le problème pouvait venir du nombre de boucles à parcourir, des calculs en flottant et de la création de variables intermédiaires dans les méthodes utilisées. Nous avons donc transformé un maximum de flottants en entiers en les multipliant par 100 si nécessaires, diminuer le nombre de boucles au maximum et mis toutes les variables de calculs en attributs de classe... Nous avons alors observé un autre problème: la taille de nos entiers. Il dépassaient parfois la valeur limite du calculateur. Il a donc fallu faire attention à ce détail. Il reste des nombres flottants, tels que a et b qui sont les coefficients de la régression linéaire (on peut se permettre ceci car ils ne sont que très peu utilisés pour des calculs). Mais le temps de calcul reste long de temps en temps (en particulier suivant la position de la ligne par rapport au bas de l'image: plus la ligne est loin, plus le calcul sera long).

De nombreuses modifications ont été apportées ensuite dans le but de pouvoir faire des tours de circuit et de diminuer le temps de calculs. Toutes nos modifications ne se sont pas avérées fructueuses mais à force de réaliser des essais, nous sommes parvenus à d'assez bons résultats.

Pour tester nos algorithmes sur circuit, nous en avons réalisé un à l'ENSICA à l'aide de ruban adhésif. Nous n'avons pas pu tester notre robot sur le circuit de la Ramée car il y avait au mois de juin des personnes utilisant la piste avec des voitures de modélisme... Cependant, notre circuit a quand même permis de tester la robustesse de nos algorithmes.

Les résultats se sont avérés plutôt positifs car nous avons réussi à faire un tour de circuit entier avec l'algorithme 1 et une moitié de circuit avec l'algorithme 2. Comme on pouvait s'en douter à la vue des premiers résultats, la vitesse du Pob-Bot est très réduite pour permettre au robot d'analyser les images et de commander la direction avant de sortir de la piste. On peut donc observer le Pob-Bot «osciller» d'une ligne à l'autre. Un problème que nous essayons encore de résoudre en ce moment et que le robot ne tourne pas brusquement en ligne droite alors qu'il ne voit pas de lignes. Nous avons pour cela des conditions sur le contraste (si le contraste est très peu élevé, alors on regarde seulement la piste) et le nombre de points utilisés pour la régression linéaire. Avec l'algorithme 1, le robot arrive à avancer droit devant pendant un instant mais il finit toujours par tourner. Cela peut aussi être dû au bitume qui n'est pas toujours très «propre» (les travaux ne laissent pas le bitume éclatant).

Il est aussi à noter que chacun des deux algorithmes met , à son lancement, un certain temps avant d'être réactif à la vue d'une ligne (une dizaine de secondes). Nous ne savons

pas encore à quoi cela est dû et essayons de résoudre cela bien que ceci n'empêche en rien le bon fonctionnement des deux algorithmes par la suite.



Figure 4.3 Piste réalisée à l'ENSICA

Chapitre 5

Conclusion

Ce projet a été intéressant et frustrant par moment. Tout d'abord, il nous a permis de découvrir les problèmes rencontrés lors d'un travail sur système embarqué. En effet, les limitations mémoire et au niveau du langage utilisé nous ont forcé à utiliser un minimum de ressources et un code le plus simple possible. Cela peut paraître facile mais quand on a l'habitude de travailler sous Eclipse sur des machines «sans limitations», il faut alors se forcer à coder différemment. C'est en ce sens que le projet nous a paru frustrant par moment. En effet, en voyant les bons résultats observés sous Eclipse avec nos algorithmes, la vue du robot tournant du mauvais côté était assez décourageante...

Il est à noter que nous n'avons tout de même pas entièrement rempli notre objectif. Le Pob-Bot devait initialement réaliser des tours de circuit à la Ramée. N'ayant pu accéder à la piste courant juin, nous ne savons pas si nos algorithmes permettent d'atteindre notre objectif de début. Notamment, la largeur de la piste à la Ramée peut engendrer des problèmes: possibilité de faire des demi-tours sur la piste. Cela vient du fait que le robot ne doit pas tourner tant qu'il ne voit pas de lignes.

Une autre approche de notre projet aurait été de coder en un langage plus proche de la machine: C ou assembleur. Peut-être que les ennuis dû à la compilation en Java et de la simplicité du langage utilisé auraient été évitées. Mais il faut d'abord passer par l'apprentissage d'un nouveau langage ce qui nécessite du temps, ce que l'on n'a pas forcément en deuxième année.

Cependant, des robots ayant une vraie plateforme Java (comme le nouveau que vient d'acquérir le DMI) peut permettre de faire beaucoup plus de choses sur un Projet d'Initiative Personnelle.

Liste des figures

Chapitre A

Annexe

A.1 ALGORITHME DE HOUGH

source: http://www.developpez.net/forums/showthread.php?t=495285

```
public class Hough {

        // image size
        private int Width,Height;

        // max Rho walue (= length of the diagonal)
        private double maxRho;

        // size of the accumulators array
        private int maxIndexTheta,maxIndexRho;

        // accumulators array
        int[][] acc;

        /**
         * Constructor
         *
         * @param width,height Size of the image
         * @param scale precision of the detection (1=best)
         */
        public Hough(int width,int height,int scale) {
                this.Width=width;
                this.Height=height;

                this.maxRho = Math.sqrt( width*width + height*height );
                this.maxIndexTheta=(int)(2*Math.PI/Math.atan2(1, Math.max(height,width)))
```

```
		this.maxIndexRho=(int)maxRho;
		this.acc = new int[maxIndexTheta][maxIndexRho];
	}

	/**
	 * pixel vote
	 *
	 * @param x,y coordinates of the pixel
	 */
	public void vote(int x,int y) {
		// for each theta value
		for(int indexTheta=0; indexTheta<maxIndexTheta; indexTheta+=1) {
			double theta = (2*Math.PI*indexTheta)/maxIndexTheta;

			// compute corresponding rho value
			double rho = x*Math.cos(theta) + y*Math.sin(theta);
			if (rho<0) continue;

			// (rho,theta) -> index
			int indexRho   = (int) (0.5 + rho/maxRho * maxIndexRho  );

			// increment accumulator
			if (indexTheta<maxIndexTheta && indexRho<maxIndexRho)
				acc[indexTheta][indexRho]++;
		}
	}

	/**
	 * retrieve winner
	 *
	 * @return array={rho,theta}
	 */
	public double[] winner() {
		// parsing the accumulators for max accumulator
		double max=0, winrho=0, wintheta=0;
		for(int r=0;r<maxIndexRho;r++) {
			for(int t=0;t<maxIndexTheta;t++) {
				if (acc[t][r]<max) continue;
				max=acc[t][r];
				winrho=r;
				wintheta=t;
			}
		}

		// index -> (rho,theta)
		double rho   = ( winrho/maxIndexRho )*maxRho;
```

```
                double theta = ( wintheta/maxIndexTheta )*2*Math.PI;

                return new double[] {rho,theta};
        }

        // convert (rho,theta) to (a,b) such that Y=a.X+b
        public double[] rhotheta_to_ab(double rho, double theta) {
                double a=0,b=0;
                if(Math.sin(theta)!=0) {
                        a = -Math.cos(theta)/Math.sin(theta);
                        b = rho/Math.sin(theta);
                } else {
                        a=Double.MAX_VALUE;
                        b=0;
                }
                return new double[] {a,b};
        }
}
```

A.2 FILTRE DE CANNY

```
 mport java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.FileOutputStream;
import javax.imageio.ImageIO;
import javax.swing.JFileChooser;

public class Canny {

private int[][] sobelX = new int[3][3] ;
private int[][] sobelY = new int[3][3] ;


public Canny(){
// on définit les matrices pour les convolutions
sobelX[0][0]= -1 ;
sobelX[0][1]= 0 ;
sobelX[0][2]= 1 ;
sobelX[1][0]= -2 ;
sobelX[1][1]= 0 ;
sobelX[1][2]= 2 ;
sobelX[2][0]= -1 ;
sobelX[2][1]= 0 ;
sobelX[2][2]= 1 ;
```

```
sobelY[0][0]= -1 ;
sobelY[0][1]= -2 ;
sobelY[0][2]= -1 ;
sobelY[1][0]= 0 ;
sobelY[1][1]= 0 ;
sobelY[1][2]= 0 ;
sobelY[2][0]= 1 ;
sobelY[2][1]= 2 ;
sobelY[2][2]= 1 ;

JFileChooser choix = new JFileChooser();
if (choix.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
String filename = choix.getSelectedFile().getAbsolutePath();
try {
BufferedImage bi = loadImage(filename);
algo(bi);
} catch (Exception e) {
e.printStackTrace();
}
}
}


public BufferedImage loadImage(String f) throws Exception {
java.io.FileInputStream in = null;
in = new java.io.FileInputStream(f);
BufferedImage input = ImageIO.read(ImageIO.createImageInputStream(in));
return input;
}


public static void main(String[] args) {
new Canny() ;
}

public void algo(BufferedImage im){
int h = im.getHeight();
int w = im.getWidth();

for(int i=3; i<w-3; i++){
for(int j=3; j<h-3; j++){

if( normePixel(im,i,j)>normePixel(im,i-1,j-1) && normePixel(im,i,j)>normePixel(im,i-1,j) && n
im.setRGB(i, j, Color.white.getRGB());
}else{
```

```
im.setRGB(i, j, Color.black.getRGB());
}

}
}

Graphics g = im.createGraphics();
g.drawImage(im, 0, 0, null);
g.dispose();
try {
FileOutputStream os = new FileOutputStream("test.png");
ImageIO.write(im, "png", os);
os.close();
im = null;
os = null;
} catch (Exception e) {
System.out.println(e);
}
}

public double sobelX(BufferedImage im, int i, int j){
double result = 0 ;
for(int n=0; n<3; n++){
for(int m=0; m<3; m++){
result+=sobelX[n][m]*(imageGris(im,i+(n-1), j+(m-1))) ;
}
}
return result ;
}

public double sobelY(BufferedImage im, int i, int j){
double result = 0 ;
for(int n=0; n<3; n++){
for(int m=0; m<3; m++){
result+=sobelY[n][m]*(imageGris(im,i+(n-1), j+(m-1))) ;
// result+=sobelY[n][m]*(imageGris(im,i+(n-1), j+(m-1))) ;
}
}
return result ;
}

// On calcule la norme d'un pixel
public double normePixel(BufferedImage im,int i,int j){
double result=0 ;
result = Math.sqrt( sobelX(im,i,j)*sobelX(im,i,j)+sobelY(im,i,j)*sobelY(im,i,j) ) ;
return result ;
```

```
}

// On donne à chaque pixel sa valeur en niveau de gris.
public double imageGris(BufferedImage im, int i, int j){
double result =0 ;
result = 0.114*(im.getRGB(i, j)& 0xff) + 0.587*((im.getRGB(i, j) >> 8)&0xff) + 0.299*((im.get
return result ;
}

}
```

A.3 ALGORITHME 1

A.3.1 Version Eclipse

Cette algorithme permet de charger une image et de récupérer l'image après traitement.

```
 import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.awt.image.BufferedImage;
import java.io.FileOutputStream;
import java.util.ArrayList;
import javax.imageio.ImageIO;
import javax.swing.JFileChooser;

public class Trait {

private ArrayList<Point> listePointsLigne;
float a;
float b;

public static void main(String[] args) {
new Trait();
}


public Trait()
{
listePointsLigne = new ArrayList<Point>() ;

JFileChooser choix = new JFileChooser();
if (choix.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
String filename = choix.getSelectedFile().getAbsolutePath();
try {
BufferedImage bi = loadImage(filename);
```

```
algo(bi);
} catch (Exception e) {
e.printStackTrace();
}
}
}


public BufferedImage loadImage(String f) throws Exception {
java.io.FileInputStream in = null;
in = new java.io.FileInputStream(f);
BufferedImage input =    ImageIO.read(ImageIO.createImageInputStream(in));
return input;
}


public void algo(BufferedImage im)
{
int h = im.getHeight();
int w = im.getWidth();
int max = 0;
int min = 2*255 ;
int nbBlanc=0;

// calcul du max et du min
for (int m =0; m<w; m++)
{
for (int n =(h/2); n<h; n++)
{
int t = ((im.getRGB(m, n)& 0xff) + ((im.getRGB(m, n) >> 8)&0xff)) ;//+ ((im.getRGB(m, n)
if (t > max)
{
max = t;
}
if(t<min){
min = t ;
}
}
}

System.out.println("max : "+max);
System.out.println("min : "+min);
float q = ((float) (max-min))/((float) max);
System.out.println("q : "+q);
System.out.println(0.9-0.2*q*q);
```

```
// traitement de l'image avec un filtre
for (int m =0; m<w; m++)
{
for (int n =(h/2); n<h; n++)
{
if(q<0.74){
im.setRGB(m, n, Color.black.getRGB());
}else{
if ( ((im.getRGB(m, n)& 0xff) + ((im.getRGB(m, n) >> 8)&0xff)) >((1-0.3*q*q)*max))  //+ ((im.
{
im.setRGB(m, n, Color.white.getRGB());
nbBlanc++;
}else{
im.setRGB(m, n, Color.black.getRGB());
}
}
}
}

// on re traite l'image
for (int m =1; m<w-1; m++)
{
for (int n =(h/2)+1; n<h-1; n++)
{
if(voisinBlanc(im,m,n)){
im.setRGB(m, n, Color.black.getRGB()) ;
}
}
}

// on regarde la proportion de blancs
float proportion = (float) (4*nbBlanc)/(h*w) ;
System.out.println("proportion : "+proportion);

if(proportion>0.3){
for (int m =0; m<w; m++)
{
for (int n =(h/2); n<h; n++)
{
im.setRGB(m, n, Color.black.getRGB());
}
}
}

// on calcule les points moyens
calculPointRouge(im) ;
```

```
regressionLineaire();
regresseionLineaireIterative();

//tracer la droite
for (int k =0; k<w; k++)
{
float y=b+a*((float)k);
if (0<=y && y<h)
{
im.setRGB(k,(int)y,Color.blue.getRGB());
}
}

if( (b<h/2+15)&(a<0.1) ){
System.out.println("Ne rien faire") ;
}

Graphics g = im.createGraphics();
g.drawImage(im, 0, 0, null);
g.dispose();
try {
FileOutputStream os = new FileOutputStream("test.png");
ImageIO.write(im, "png", os);
os.close();
im = null;
os = null;
} catch (Exception e) {
System.out.println(e);
}
}


public void calculPointRouge(BufferedImage im){
int h = im.getHeight();
int w = im.getWidth();

for (int m =0; m<w; m++)
{
int nb=0;
int position=0;
int debut=0;
for (int n =(h/2); n<h; n++)
{
if(im.getRGB(m, n)==0xffffffff)
{
```

```
nb++;
position=0;
if(debut==0)
{
debut=n;
}
}
else if(nb!=0&&position<2)
{
position++;
}
else if(nb>1)
{
listePointsLigne.add(new Point(m,n-position-(n-debut)/2));
im.setRGB(m, n-position-(n-debut)/2, Color.RED.getRGB());
nb=0;
position=0;
debut=0;
}
else
{
nb=0;
position=0;
debut=0;
}
}
}
}


public boolean voisinBlanc(BufferedImage im,int n,int m){
boolean bool = false ;
int compteur = -1 ;
for(int i=-1; i<2; i++){
for (int j=-1 ;j<2; j++) {
if( im.getRGB(n+i, m+j)==Color.white.getRGB() ){
compteur++;
}
}
}
if(compteur<3){
bool = true;
}
return bool ;
}
```

```
public void regressionLineaire(){
int n =0;
int sumxx=0;
int sumxy=0;
int sumx = 0;
int sumy =0 ;
float Sxx, Sxy;

for (Point p:listePointsLigne){
n++;
sumx += (float)p.getX();
sumy += (float)p.getY();
sumxx += ((float)p.getX())*((float)p.getX());
sumxy += ((float)p.getX())*((float)p.getY());
}
Sxx = sumxx-sumx*sumx/((float)n);
Sxy = sumxy-sumx*sumy/((float)n);
a =Sxy/Sxx;
b = (sumy-a*sumx)/((float)n);
System.out.println("a : "+a+" , b : "+b) ;
}


public void regresseionLineaireIterative(){
ArrayList<Point> nouvelleListePointsLigne= new ArrayList<Point>();

for(Point p : listePointsLigne)
{
if(Math.abs(((float)p.getY())-(b+a*((float)p.getX())))<8)
{
nouvelleListePointsLigne.add(p);
}
}
listePointsLigne=nouvelleListePointsLigne;
regressionLineaire();
}

}
```

A.3.2 Version Pob-Bot

Voici un code pour l'algorithme 1 qui a permis de faire des tours sur le circuit de l'ENSICA. Ce code nécessite l'utilisation de la classe Bot fourni dans la documentation du Pob-Bot et aussi donné en fin d'annexe.

```
 import com.pobtechnology.pobeye.* ;
//import com.pobtechnology.poblcd.* ;
//import java.lang.RuntimeException;

class Main
{

// essayez de virer au maximum les attributs que l'on utilise pas en parallele et qui sont de
int h = 120;
int w = 88;
private int [] listePointsLignex;
private int [] listePointsLigney;
private int indice;
private int max;
private int min;
private int nbBlanc;
private int q;
//private int t;
private int valeurPixel;
private int proportion;
private float a;
private float b;
private int nb;
private int position;
private int debut;
// attribut regression lineaire
private int n ;
private int sumxx;
private int sumxy;
private int sumx ;
private int sumy ;
private float Sxx;
private float Sxy;
private int xreg ;
private int yreg ;
private int indiceEcriture;

 public Main()
 {
listePointsLignex=new int[172];
listePointsLigney=new int[172];

for(int i = 0;i<172;i++)
{
listePointsLignex[i]=0;
listePointsLigney[i]=0;
```

```
}

// pob-proto board init
Bot.init();
RGBFrame.init();

Bot.setHead(160);
System.wait(100000);
Bot.setHead(120);
System.wait(100000);
Bot.setHead(200);
System.wait(100000);
Bot.setHead(160);

// joystick init
//Pad.init();

// main loop
while(true)
{
algo();
if(indice>3)
{
if (a>0.01f)
{
Bot.setHead(110);
}else{
if(a<-0.01f)
{
Bot.setHead(210);
}
else
{
Bot.setHead(160);
}
}
}
else
{
Bot.setHead(160);
}
}

 }

public static void main(String []args)
```

```
{
new Main();
}

public void algo()
{
RGBFrame.grab();
max = 0;
min = 2*255 ;
nbBlanc=0;

// calcul du max et du min
for (int y =60; y<h; y++)
{
for (int x =0; x<w; x++)
{
valeurPixel = RGBFrame.getGreenPixel(88*y+x)+ RGBFrame.getBluePixel(88*y+x);
if (valeurPixel > max)
{
max = valeurPixel;
}
if(valeurPixel<min){
min = valeurPixel ;
}
}
}

q = ( (100*(max-min))/max );

indice=0;

// traitement de l'image avec un filtre et acquisition des points de la ligne en simultanée
if(q>70)
{
for (int x =0; x<w; x++)
{
nb=0;
position=0;
debut=0;

for (int y =60; y<h; y++)
{
valeurPixel=RGBFrame.getGreenPixel(88*y+x)+ RGBFrame.getBluePixel(88*y+x);

if ( (20*valeurPixel) >  (17*max) )
{
```

```
nbBlanc++;
nb++;
position=0;
if(debut==0)
{
debut=y;
}
}
else{
if( (nb!=0) && (position<2) )
{
position++;
}
else{
if((nb>1)&&(indice<172))
{
listePointsLignex[indice]=x;
listePointsLigney[indice]=y-position-((y-debut)/2);
indice++;
nb=0;
position=0;
debut=0;
}
else{
nb=0;
position=0;
debut=0;
}
}
}
}
}

// on regarde la proportion de blancs
proportion = ( (200*nbBlanc)/(h*w) ) ;
if((proportion>40)){
a=0f;
b=0f;
}
else{
regressionLineaire();
if(indice>4)
{
regresseionLineaireIterative();
}
else
```

```
{
a=0f;
b=0f;
}
}
}
else
{
a=0f;
b=0f;
}

}

public void regressionLineaire(){
n =0;
sumxx=0;
sumxy=0;
sumx = 0;
sumy =0 ;
Sxx=0f;
Sxy=0f;

while (listePointsLigney[n]!=0 && n<indice){
sumx += listePointsLignex[n];
sumy += listePointsLigney[n];
sumxx += (listePointsLignex[n]*listePointsLignex[n]);
sumxy += (listePointsLignex[n]*listePointsLigney[n]);
n++;
}
Sxx = (((float)sumxx)-(((float)(sumx*sumx))/((float)indice)));
Sxy = (((float)sumxy)-(((float)(sumx*sumy))/((float)indice)));
a =((Sxy)/(Sxx));
b = ((((float)(sumy))-(a*((float)sumx)))/((float)indice));
}

public void regresseionLineaireIterative(){
indiceEcriture=0;

for(int indiceLecture=0;indiceLecture<indice;indiceLecture++)
{
xreg =listePointsLignex[indiceLecture];
yreg =listePointsLigney[indiceLecture];
if(  ((((float)yreg)-b-(a*((float)(xreg))) )<6f ) && ( (b+(a*((float)xreg))-((float)(yreg)))
{
listePointsLignex[indiceEcriture]=xreg;
```

```
listePointsLigney[indiceEcriture]=yreg;
indiceEcriture++;
}

}
indice=indiceEcriture;
regressionLineaire();
}
}
```

A.4 ALGORITHME 2

A.4.1 Version Eclipse

Cette algorithme permet de charger une image et de la traiter avec le deuxième algorithme. L'image obtenue est renvoyée dans une fenêtre avec l'image d'origine pour pouvoir analyser les résultats.

```
 import java.awt.BorderLayout;
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.BufferedImage;
import java.io.FileOutputStream;
import java.util.Iterator;
import javax.imageio.ImageIO;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class Trait extends JPanel implements MouseListener, ActionListener{

private BufferedImage im, im2;
private JLabel label;

public Trait()
{
JFileChooser choix = new JFileChooser();
```

```
if (choix.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
String filename = choix.getSelectedFile().getAbsolutePath();
try {
JFrame frame = new JFrame();

im = loadImage(filename);
im2 = loadImage(filename);
frame.getContentPane().add(this);
this.setLayout(new BorderLayout());
this.addMouseListener(this);
JButton button = new JButton("Calcul");
button.addActionListener(this);
button.setActionCommand("c");
this.add(button, BorderLayout.SOUTH);
frame.getContentPane().add(this);
frame.setPreferredSize(new Dimension(300,300));
frame.pack();

frame.setVisible(true);
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
}

public void paint(Graphics g)
{
g.drawImage(im2, 106, 0, null);
g.drawImage(im, 106, 120, null);
}

public BufferedImage loadImage(String f) throws Exception {
Image im2 = null;
java.awt.MediaTracker mt2 = null;

java.io.FileInputStream in = null;
byte[] b = null;
int size = 0;

in = new java.io.FileInputStream(f);
BufferedImage input = ImageIO.read(ImageIO.createImageInputStream(in));

return input;
}
```

```
public void algo(BufferedImage im)
{
int h = im.getHeight();
int w = im.getWidth();
int max = 0;
int min = 0;
for (int m =0; m<w; m++)
{
for (int n =0; n<h; n++)
{
int t = valMoy(im2,m,n,2);
if (t > max)
{
max = t;
}
if (t <min)
{
min = t;
}
}
}
int ecart = max - min;
System.out.println("Ecart : "+ecart);

int ech = valMoy(im, w/2, 7*h/8, 5);
System.out.println(ech);

int limHG = 0;
int limHD = w-1;
int limV = 0;

int[][] result = new int[2][88];

for (int i=w-1; i>=0; i--)
{
int q = h-1;
boolean b = true;
while (q>0 && b)
{
if (Math.abs(valMoy(im2,i,q,5)-ech)<10)
{
im.setRGB(i, q, Color.yellow.getRGB());
}else{
b = false;
}
q--;
```

```
}

im.setRGB(i, q+1, Color.red.getRGB());
result[0][i] = i;
result[1][i] = q+1;
}

interpolation(result);

Graphics g = im.createGraphics();
g.drawImage(im, 0, 0, null);
g.dispose();
try {
FileOutputStream os = new FileOutputStream("test.png");
ImageIO.write(im, "png", os);
os.close();
im = null;
os = null;
} catch (Exception e) {
// TODO: handle exception
System.out.println(e);
}

}

public int valMoy(BufferedImage im, int x, int y, int rayon)
{
int tot = 0;
int compt =0;
for (int i = (x-rayon); i<(x+rayon); i++)
{
for (int j = (y-rayon); j<(y+rayon); j++)
{
if (i>=0 && i<im.getWidth() && j>=0 && j<im.getHeight())
{
tot = tot +  ((((im.getRGB(i, j) >> 8)&0xff) + ((im.getRGB(i, j))&0xff))/2);
compt++;
}

}
}
if (compt != 0) tot = tot/compt;
return tot;
}

public int[][] pointsLimites()
```

```
{
int[][] result = new int[2][88];
for (int i = 0; i<im.getWidth(); i++)
{
boolean b = true;
int j = im.getHeight()-1;
while (im.getRGB(i, j) == Color.yellow.getRGB() && b)
{
j--;
}
im2.setRGB(i, j, Color.red.getRGB());
result[0][i] = i;
result[1][i] = j;
}
return result;
}

public float[] interpolation(int[][] points)
{
float sxy = 0;
for (int i=0; i<88; i++)
{
sxy = sxy + points[0][i]*points[1][i];
}
System.out.println(sxy);
float sx = 0;
for (int i=0; i<88; i++)
{
sx = sx + points[0][i];
}
System.out.println(sx);
float sy = 0;
for (int i=0; i<88; i++)
{
sy = sy + points[1][i];
}
System.out.println(sy);
float sx2 = 0;
for (int i=0; i<88; i++)
{
sx2 = sx2 + points[0][i]*points[0][i];
}
System.out.println(sx2);
float b1 = (sxy - sx*sy/88)/(sx2-sx*sx/88);
float b0 = (sxy-b1*sx2)/sx;
```

```
float [] result = new float[2];
result[0] = b0;
result[1] = b1;
System.out.println(""+b0+"+"+b1+"*x");

for (int i=0; i<88; i++)
{
int y = Math.round(b0+b1*i);
im2.setRGB(i, y, Color.blue.getRGB());
}

return result;
}

public void mouseClicked(MouseEvent e) {
int x = e.getX();
int y = e.getY();
System.out.println(valMoy(im2,x-106,y,3));
}

public void mouseEntered(MouseEvent e) {
}

public void mouseExited(MouseEvent e) {
}

public void mousePressed(MouseEvent e) {
}

public void mouseReleased(MouseEvent e) {
}

public void actionPerformed(ActionEvent e) {
if (e.getActionCommand().equals("c"))
{
algo(im);
this.repaint();
}
}

public static void main(String[] args) {
new Trait();
}

}
```

A.4.2 Version Pob-Bot

Algorithme qui a permis de faire une moitié de circuit. Toujours à utiliser avec la classe Bot.

```
 import com.pobtechnology.pobeye.* ;
import com.pobtechnology.poblcd.* ;

class Main
{
private int [] resultY;
private int indice;
private int valMoy, ech, tot, compt1, compt2;
private float a;
private float b;
private boolean bool;
private int h = 120;
private int w = 88;
private int x,y;
private int sumxx;
private int sumxy;
private int sumx;
private int sumy ;
private float Sxx, Sxy;


 public Main()
 {
 valMoy = 0;
 ech = 0;

// pob-proto board init
Bot.init();
RGBFrame.init();

Bot.setHead(160);
System.wait(100000);
Bot.setHead(120);
System.wait(100000);
Bot.setHead(200);
System.wait(100000);
Bot.setHead(160);

resultY = new int[88];

// main loop
while(true)
```

```
{
algo();
}
 }

public static void main(String []args)
{
new Main();
}

public void algo()
{
RGBFrame.grab();

/* Calcul de l'échantillon */
valMoy(35,100,4);
ech = valMoy;
valMoy(55,100,4);
ech = ((ech+valMoy)/2);

/* Initialisation du compteur de points valides pour l'interpolation */
compt2=0;

/* Boucle d'analyse des points */
int ybis;
for (int xbis =0; xbis<w; xbis++)
{
ybis = 120;
while (ybis>40){
valMoy(xbis,ybis,2);
if ((valMoy-ech)<40)
{
ybis = ybis-2;
}else{
resultY[xbis] = ybis;
ybis=0;
compt2++;
}
}
}

/* Test sur le nombre de points valides minimum pour la régression linéaire */
if (compt2>15)
{
regressionLineaire();
```

```
if ((a>0f)&&(b<100))
{
Bot.setHead(110);
}else{
if((a<0f)&&(b>60))
{
Bot.setHead(210);
}else{
Bot.setHead(160);
}
}
}else{
Bot.setHead(160);
}


}

public void regressionLineaire(){
 sumxx = 0;
 sumxy = 0;
 sumx = 0;
 sumy = 0;

for(int i=0; i<88; i++)
{
y =resultY[i];
if(y > 40)
{
x =i;
sumx += x;
sumy += y;
sumxx += ((x)*(x));
sumxy += ((x)*(y));
}
}
Sxx = (((float)sumxx)-(((float)(sumx*sumx))/(88f)));
Sxy = (((float)sumxy)-(((float)sumx*sumy)/(88f)));
a =(Sxy/Sxx);
b = ((((float)sumy)-(a*((float)sumx)))/(88f));
}

public void valMoy(int xbis, int ybis, int rayon)
{
tot = 0;
compt1 =0;
```

```
for (int i = (xbis-rayon); i<(xbis+rayon); i++)
{
for (int j = (ybis-rayon); j<(ybis+rayon); j++)
{
if (i>=0 && i<88 && j>=0 && j<120)
{
tot = (tot +  (RGBFrame.getGreenPixel(88*ybis+xbis)+ RGBFrame.getBluePixel(88*ybis+xbis)));
compt1++;
}
}
}
if (compt1 != 0) {tot = tot/compt1;}
valMoy = tot;
}
}
```

A.5 LIENS UTILES SUR LA RECONNAISSANCE DE LIGNES, CONTOURS

http://emmanuel.harel.free.fr/
http://www.developpez.net/forums/showthread.php?t=495285